2. Séquence numéro 2 : modèle mémoire et surcharge de méthode. 2h CM, 2h TD, 4h TP

2.1. TD - attributs et méthodes statiques.

2.1.1. Exercice 1

2.2. TP - initialisation de notre petit projet

Le but de l'ensemble du TP est de programmer un petit moteur de jeu permettant d'afficher un dongeon de manière statique (sans caméra) et des sprites pouvant se mouvoir dans ce dongeon en respectant une physique élémentaire (gestion des collisions au minimum). Beaucoup des classes que nous allons développer existe déjà quelque part, y compris dans les classes élémentaires du coeur de Java (par exemple, notre classe Hitbox va beaucoup ressembler à la classe Rectangle2D). Cependant, le but est pédagogique, ce qui fait que nous ne prendrons pas toujours le chemin le plus court.

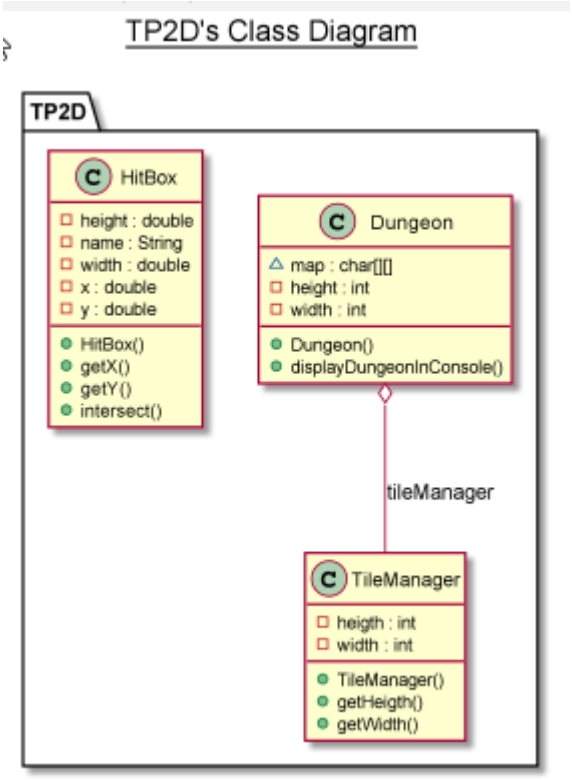

Figure 2.1.: Nos premières classes de TP.

2.2.1. La classe Hitbox

La classe Hitbox permet de modéliser les collisions entre des éléments statiques (murs, pièges etc) et des éléments dynamiques (le héro, des antagonistes etc), ou bien entre éléments dynamiques.

Dans cette première partie, HitBox permet juste de modéliser un rectangle dont le coin en haut et à gauche est en position (x,y) (x est orientée vers la droite et y vers le bas, comme toujours en informatique 2D), de la largeur width (sur l'axe x) et de hauteur height (sur l'axe y).

Dans un premier temps, on souhaite être capable de vérifier si un objet HitBox est par dessus une autre Hitbox. C'est la méthode `boolean intersect(HitBox anotherHitBox);`.

Pour écrire cette méthode, on va tout d'abord déterminer le rectangle le plus à gauche. Une variable xOverlap vaut 1 si la largeur du rectangle le plus à gauche est supérieure à la distance sur l'axe x entre les deux rectangles.

De la même manière, on détermine la rectangle le plus haut. Une variable yOverlap vaut 1 si la hauteur de ce rectangle est plus haute que la distance sur l'axe y entre les deux rectangle.

Il y a intersection lorsque ces deux variables booléennes valent 1.

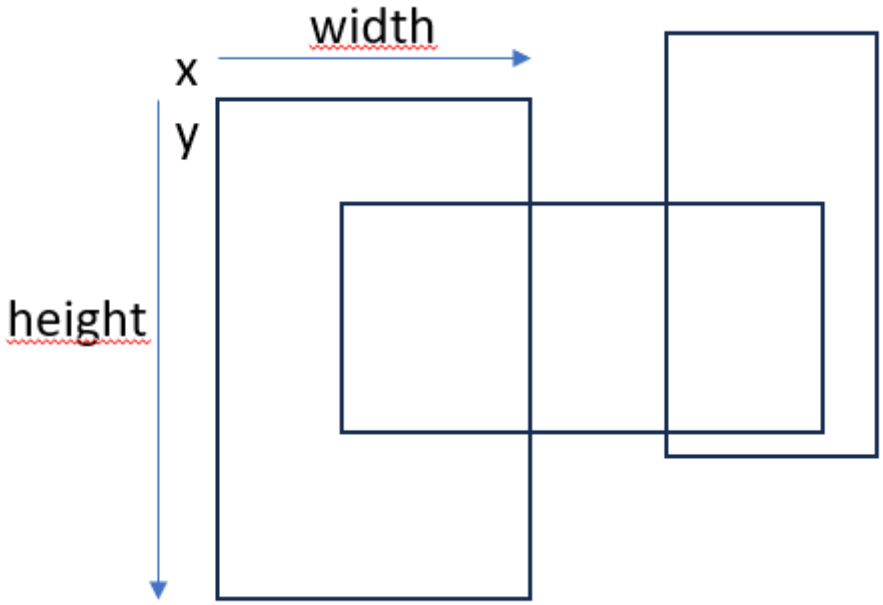

Figure 2.2.: Intersection de HitBoxes.

- Codez la classe HitBox avec ses paramètre et ses méthodes.
- Codez des tests dans une classe Test permettant de vérifier notamment la méthode intersect.

2.2.2. La classe TileManager

Cette classe sera à terme relativement complexe, mais mpour le moment elle comporte juste deux attributs privé et finaux, la hauteur et la largeur en pixel de chaque tuile du jeux.

Cette classe permet de charger les tuiles puis d'accéder à ces images.

- Codez la classe TileManager ayant deux paramètres finaux : height et width.
- La classe possède un constructeur simple.
- La classe possède deux getter pour height et width.

2.2.3. La classe Dungeon

Cette classe permet de modéliser un dongeon. Elle est composé pour le moment de quatre attributs : un tableau à deux entrée contenant des caractères, une hauteur et une largeur (finaux), et un objet de type TileManager.

- Codez la classe Dungeon.
- La classe possède un constructeur simple. Ce constructeur initialise le tableau du dongeons de la manière suivante : sur les bord du tableau, le constructeur initialise avec un W (pour Wall), à l'intérieur du tableau, le constructeur initialise avec le caractère ' ' (espace).
- La méthode `void displayDungeonInConsole(HitBox hero);` affiche dans la console le tableau du dongeons, en mettant un "H" à la position du héro. On connait la position du héros via la largeur et la hauteur des tuiles.

2.2.4. La cascade d'héritage permettant de gérer les affichages

On cherche maintenant à implémenter une série d'objet héritant les uns des autres. Voici leur utilité :

1. La classe Things permet de modéliser trucs : les éléments intangibles du décors à afficher (notamment le sol). Ils sont caractérisé par une position (x,y) et une taille (widht, height).
2. La classe SolidThings permet de modéliser les éléments ayant une consistance : coffre, mur, obstacles divers etc. Elle possède donc une référence vers une HitBox.
3. La classe AnimatedThings permet de modéliser les éléments ayant une animation, par exemple une torche.
4. La classe DynamicThings permet de modéliser les éléments ayant une physique. Ils peuvent se déplacer tout en étant animé. Ils possèdent entre autre une direction.

Le schéma ci-dessous montre cette cascade d'héritage.

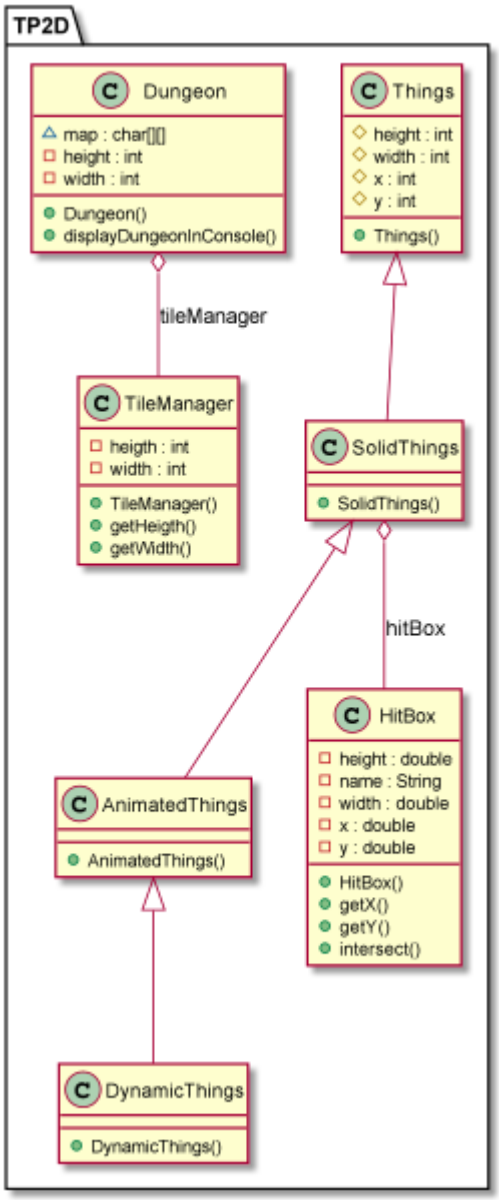

Figure 2.3.: La cascade d'héritage des classes Things.

- Codez la classe Things avec son constructeur.
- Codez les classses SolidThings et son constructeur. En l'absence d'une précision, la taille de la hitBox est la même que la taille de Things.
- Codez les classes AnimatedThings et DynamicThings. La classe dynamicThings comprend deux autres variables : speedX et speedY.

2.2.5. Modification de la classe Dungeon

- Au sein de la classe Dungeon, ajouter un tableau dynamic de Things représentant l'ensemble du donjons.
- Codez une méthode fillThingsArray qui génère la liste des Things et des SolidThings en fonction de la variable map. Pour chaque caractère ' ' (espace, solide), on ajoute une instance de Things, pour chaque caractère 'W' (wall) on ajoute une instance de SolidThings.
- Vérifiez avec le débugger.